DISTRIBUTED APPLICATION CONTROL SYSTEM, CONTROL METHOD AND A PROGRAM

FIELD OF THE INVENTION

This invention relates to an information processing system and, more particularly, to a distributed application control system, employing an agent, a control method and a program product carryable by a recording medium.

As will be apparent from the following explanation, the present invention relates to a technique applied with advantage to a distributed application control system in which plural files distributed in plural computers are supervised in a unified or systematic directory and in which a movable control agent system moves or communicates an application agent supervising an application executing services supplied by each file to a computer where there exists the pertinent file, to execute a script to execute application control.

BACKGROUND OF THE INVENTION

For example, the NFS (Network File System) of Sun Microsystems Inc. provides such a system for plural computers interconnected over a network, in which, in a well-known manner, a file system of a remote computer is caused to appear as if it is a local computer file system for a directory supervision so that the computer can access a file of the remote computer in the same way as the local file system. However, the file is processed by a local computer, such that file

processing cannot be carried out on the remote computer.

Moreover, a remote shell mounted on a UNIX system is a script language configured for executing commands on a remote computer, as described in Document 1 (Unix Network Programming, W. Richard Stevens, Prentice-Hall, 1990). In this script language, the executing computer is specified and the command specified by the script is executed on the specified computer, providing, however, that one script is interpreted on only one computer.

For executing the command on plural computers, there is no alternative but to provide plural scripts each designating a computer for execution, and to invoke the plural scripts sequentially as remote shells in association with each of the plural computers.

As a technique for continuing execution across or over plural computers, there is known a movable agent system.

For example, in a telescript technology of General Magic Inc., USA, an agent moves among different computers under a "go" command, as described in Document 2 (Publication of JP Patent Kokai JP-A-7-182174. In order for an agent to have communication with another agent, the agent needs to be moved at the outset to a special site called a "place" where a counterpart agent exists. In other words, communication is possible only between agents existing in the same place. Although the telescript technique features the itineration across plural computers to receive services, it suffers a drawback that place management needs to be performed separately from service

management and that one agent cannot access plural computers in parallel.

The "Agent Tcl", also called "D' Agent", disclosed in Document 3 ("William Cockayne and Michael Zyda, Mobile Agent, Manning Publications Co., 1997"), is a mobile agent developed by Dartmouth University, USA. This mobile agent is based on a shell script language, called "Tcl" (Tool Command Language), to which is added a movement function by an "agent_jump" command. This "Tcl" can be utilized as a simple shell and represents a shell script language that can be used more readily by a user than the program language exemplified by telescript.

However, even in the "Agent Tcl", agent movement across the computers is performed by a movement command and supervised by a directory separate from the directory for the resources in the computer. The computer and the resources in the computer are supervised by separate directories, such that a management of the movement command is independent of a management of the computer resources.

On the other hand, since an agent can exist only in one computer, the resources of plural computers cannot be handled in parallel.

In, e.g., Document 4 (JP Patent Kokai JP-A-10-149287), there is disclosed a mobile agent system of the predicate logic type in which an action of the script language for causing the agent movement by a [goto] command can be stated as a predicate.

In the system described in Document 4, the script language can

describe the agent movement, however, it cannot supervise the directory together with the resources furnished by the computer. Moreover, similarly to other mobile agent languages, the script language can not access the resources of plural computers in parallel.

In, e.g., unpublished Document 5 (JP Patent Application No. 11-047015, not published as of the filing date of the present Japanese application, now Kokai Publication JP-P2000-244426A published Sept. 8, 2000), there is described a system which not only has an agent movement command and is able to move a program being executed to different computers but also has a function of permitting an agent to perform execution in parallel across plural computers such that a program unifies services present in distribution across plural computers to supervise the services in a distributed and synchronous fashion to provide a service over a network. However, commands for movement and management of the computer resources are performed independently of each other, such that a user cannot easily access computer resources distributed over the network. However, there is much to be desired in this system, likewise other techniques disclosed by the published documents.

SUMMARY OF THE DISCLOSURE

In the course of eager investigations towards the present invention, it has turned out that the techniques described in the above published documents suffer the following problems.

A first problem is that the above-described techniques fail to

provide a control environment in which plural remote computers and the resources furnished by these remote computers are supervised by a simplified directory structure and in which the directory is moved to generate an agent supervising the resources in a site where there exist these resources to enable interactive control of resources.

The reason is that a system not employing a mobile agent lacks in a function of facilitated movement across plural computers such that an agent for managing the resources cannot be prepared on a remote computer.

If the system utilizes a mobile agent, limitations may be imposed on communication with remote computers, or movement across different computers and the directory management of the computer resources are managed separately.

The second problem is that, in a conventional mobile agent system, means for sequentially controlling services presented by plural remote computers from a control agent and means for distributedly and synchronously controlling the services cannot be provided together or simultaneously.

The reason is that the mobile agent systems described in the above-mentioned several published documents are provided with a function of sequentially itinerating through plural computers, however, are not provided with a function of having plural services controlled by an agent existing in a distributed fashion across plural computers.

If an agent system is provided with a function for movement and

a function for distribution, it is targeted to construct a program language for developing an agent system, whereas it is not targeted to provide a script language aimed at permitting facilitated combination of sequential control and distributed control by a user.

In view of these problems, it is an aspect of the present invention to provide a system, method and program product for readily controlling, through use of a script language, the generation, movement and the end of an agent, configured for supervising resources on plural computers, by a user interactive environment.

It is another aspect of the present invention to provide a system, method and program product in which a system for controlling an agent on a remote controller is provided with a movement function and a distribution function and in which computer movement and management of resources are caused to appear as a sole directory structure to the user, whereby even a user not well versed in computer networks is able to readily state by a script language an agent realizing a unified service in need of the movement function and the distribution function.

Other aspects, objects, features and advantages of the present invention will become apparent to those skilled in the art from the following description and the claims.

According to an aspect of the present invention, there is provided a system comprising a shell agent responsive to an input of a script language configured for controlling distributed application to interpret and execute the script language, a local

service agent furnishing information of a local file system to a computer, and an application agent directly controlling the application, an agent base furnishing respective fields of execution to the shell agent, the local service agent and the application agent. The system further comprises agent mover for causing movement of an agent to an agent base of at least one other computer, remote call module for furnishing a function for an agent to have communication with an agent of an own computer or at least one other computer, and agent generator for generating an application agent. The execution of the application distributed over each computer is controlled responsive to an input of the script language.

According to a second aspect of the present invention, there is provided a distributed application control method in which a shell agent in each agent platform of each computer interprets and executes a script language input to control the distributed application, a local service agent furnishes information pertinent to a local file system to a computer, and an application agent directly controls the application. An agent base furnishes respective fields of execution for the shell agent, the local service agent and the application agent. An agent is movable to at least one other agent base of at least one other computer (through an agent movement mechanism). The agent has communication with an own computer or at least one other computer through a remote call function. The shell agent interprets the input script language to control the generation of the application agent through an agent generating function. The execution of the

distributed applications in the computers is controlled responsive to inputting of the script language.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 shows a system configuration of an embodiment of the present invention.

Fig. 2 shows an agent platform according to an embodiment of the present invention.

Fig. 3 shows an illustrative structure of a shell agent according to an embodiment of the present invention.

Fig. 4 shows another illustrative structure of a shell agent according to an embodiment of the present invention.

Fig. 5 shows an illustrative structure of a directory supervising table according to an embodiment of the present invention.

Fig. 6 shows an illustrative structure of a status table according to an embodiment of the present invention.

Fig. 7 shows an illustrative structure of a shell variable supervising table according to an embodiment of the present invention.

Fig. 8 shows an illustrative structure of an agent referencing table according to an embodiment of the present invention.

Fig. 9 is a flow diagram for illustrating the processing sequence in interpreting a script language according to an embodiment of the present invention.

Fig. 10 is a flow diagram for illustrating the processing

sequence in interpreting an internal command according to an embodiment of the present invention.

Fig. 11 illustrates directory movement according to an embodiment of the present invention.

Fig. 12 illustrates a first example of directory movement according to an embodiment of the present invention.

Fig. 13 illustrates a second example of directory movement according to an embodiment of the present invention.

Fig. 14 illustrates execution of a remote command according to an embodiment of the present invention.

Fig. 15 illustrates a first example of execution of a remote command according to an embodiment of the present invention.

Fig. 16 illustrates a second example of execution of a remote command according to an embodiment of the present invention.

Fig. 17 is a flowchart for interpreting parallel execution according to an embodiment of the present invention.

Fig. 18 illustrates a flowchart showing a first example of parallel execution according to an embodiment of the present invention.

Fig. 19 illustrates a flowchart showing a second example of parallel execution according to an embodiment of the present invention.

Fig. 20 is a flowchart showing a processing sequence for interpreting execution of an application agent according to an embodiment of the present invention.

Fig. 21 illustrates a flowchart showing a first example of execution of an application agent according to an embodiment of the present invention.

Fig. 22 illustrates a flowchart showing a processing sequence for interpreting execution of an application agent according to an embodiment of the present invention.

Fig. 23 is a flowchart showing a second example of execution of an application agent according to an embodiment of the present invention.

Fig. 24 shows an illustrative relation between an application agent and an application according to an embodiment of the present invention.

Fig. 25 shows an example of the relation between a general-purpose application agent and the application according to an embodiment of the present invention.

Fig. 26 illustrates a first embodiment of a system embodying the present invention.

Fig. 27 illustrates a second embodiment of a system embodying the present invention.

Fig. 28 illustrates a third embodiment of a system embodying the present invention.

PREFERRED EMBODIMENTS OF THE INVENTION

A preferred embodiment of the present invention is hereinafter explained. In a distributed application control system, according to an embodiment of the present invention, a shell agent interpreting

the script language moves itself or has communication with an agent platform of at least one remote computer to enable the application distributed over plural computers to be controlled by a sole script.

More specifically, Referring to Fig. 2, each computer has agent platform (2). The agent platform has an agent base (9), agent movement module (mover) (13) for furnishing the function of causing the movement of an agent to an agent base of another computer, remote call module (15) for furnishing the functions for an agent to have communication with an agent of an own computer or at least one other computer, and agent generating module (agent generator 16) for generating an application agent. The agent base includes a shell agent (10) responsive to an input of a script language configured for controlling the distributed application to interpret and execute the script language, a local service agent (11) for furnishing information of a local file system to the computer, and an application agent (12) for directly controlling the application (18). The script language input through an input module (7) is interpreted by the shell agent (10) to boot the application agent (12) which supervises an actual application (18). The shell agent and the application agent are movable through agent mover (13) to at least one other computer and is able to have communication with the other computer or computers using remote caller (15).

The shell agent includes a shell interpreter (20 of Fig. 3) and a current directory (24 of Fig. 3).

The shell agent includes a repository path table (31 of Fig. 3)

and retrieves the application agent from the repository path table to execute a retrieved application agent.

For generating an application agent at remote computer by the shell agent, there is provided means (module 30 of Fig. 4) invoking a local service agent (11 of Fig. 4) assisting in accessing resources local to the respective computers. When generating the application agent, the shell agent generates the application agent through the local service agent (11) without directly invoking the agent generating module (agent generator 16).

In an alternative embodiment of the present invention, a shell interpreter interprets a keyword [parallel] and generates a thread and a sub-shell agent to the remote computer or computers in order to execute plural commands in parallel. The sub-shell agent causes the actual application agent to be executed by the remote computer.

More specifically, the shell interpreter (20 of Fig. 3) includes syntax analyzing module (syntax analyzer 21 of Fig. 3) and causes the remote computer to generate a sub-shell agent (51 of Fig. 18) using thread generator (14 of Fig. 3) and remote caller (15 of Fig. 18).

If there is a syntax specifying parallel execution ([parallel sentence]) in the input script language, as many threads as necessary in executing the script language are generated to control the parallel execution of the application.

The shell agent includes a status table (23 of Fig. 3) for supervising the status of the agent. The status table includes

parallel execution counts of the agent and a terminal execution flag. If the current base of the current directory is a remote computer, and the status of the agent is not the parallel executing state nor the terminal executing mode, the shell agent is moved to the remote computer or computers.

When the agent platform generates an application agent in a remote computer by the generated thread, there is a sub-shell agent in the agent base in the remote computer to take over the generation of the application agent.

In another embodiment of the present invention, the application agent booted executes a specified application program dependent on the computer or employs a general-purpose application agent to selectively execute an application depending on the file.

More specifically, the application agent (12 of Fig. 24) is associated with a specified application (18 of Fig. 24) or an extension application accommodating table (98 of Fig. 25) is used as a general-purpose application agent (97 of Fig. 25) to determine the application (18 of Fig. 25).

The shell interpreter (20 of Fig. 3) of the shell agent references the information of the current directory (24 of Fig. 3) to interpret and execute the script language and invokes the agent generator (16 of Fig. 3) for generating an application agent configured for directly controlling the application and remote call module (15 of Fig. 3) configured in turn for causing movement of the shell agent itself by the agent mover (13 of Fig. 3) and providing for

communication with the remote computer as necessary.

The application agent executes the file to be executed using a specified application. Alternatively, the general-purpose application agent uses the extension application accommodating table, to determine and execute the application.

Meanwhile, the functions of the shell agent, local service agent, application agent, agent mover, thread generator, remote call module, agent generator, syntax analyzer and command analyzer-executor of a shell agent can be realized by executing the program on the computer. The present invention can be practiced by loading and installing these programs from a well-known dynamic or static medium, such as a recording medium or a communication medium, through given interfacing means, and by executing the execution image.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

For explaining the above-described preferred embodiment of the present invention in more detail, the embodiments of the present invention are explained with reference to the drawings.

Fig. 1 shows a system configuration according to an embodiment of the present invention, and illustrates a structure of a distributed processing system comprising plural computers interconnected over a network and which are configured for executing a network script language. Although Fig. 1 shows a system comprised of three computers, by way of an example, Fig. 1 and the following figures are intended for illustrating the present invention without limiting the invention. For example, the present invention is not

limited to a configuration with three computers.

Referring to Fig. 1, the computers 1A to 1C are provided with agent platforms 2A to 2C, respectively. The agent platforms are provided respectively with agent bases 9A to 9C. In these agent bases 9A to 9C are provided with agents 3A to 3C, respectively.

The computers 1A to 1C are interconnected over a network 4. Using this network 4, communication between different agent platforms or between different agents or migration of the agents are performed.

Fig. 2 shows an illustrative structure of a computer executing network script language in an embodiment of the present invention.

Referring to Fig. 2, the computer 1 has an input 5 and an output 6. In addition, the computer 1 includes an agent platform 2, a local file system 17, an application 18 and an agent repository 19.

The agent platform 2 has input module 7 for receiving the input 5 to the computer 1 and output module 8 for the output 6. The agent platform 2 also includes an agent base 9 as an area in which to store a variety of running agent programs, such as a shell agent 10, a local service agent 11, or an application agent 12.

The agent platform 2 also includes agent movement module (agent mover) 13, thread generating module 14, remote call module 15 and agent generating module 16, these modules having the functions of assisting in the agent execution.

The agent movement module 13 furnishes the function of allowing an agent in the agent base 9 to migrate to another agent platform.

The thread generating module 14 furnishes the function of generating a new thread when an agent in the agent base 9 performs multi-thread operations.

The remote call module 15 furnishes the function for an agent in the agent base 9 to cause another agent in the same agent platform or an agent in another agent platform to invoke a method.

The agent generating module 16 is invoked when a new agent is generated and executed in the agent base 9. Specifically, the agent generating module 16 retrieves an agent program in the agent repository 19 and, based on the retrieved result, generates an agent in the agent base 9.

The shell agent 10 is an agent which interprets and executes a network script language and performs communication with a local service agent 11, while generating and having communication with the application agent 12.

The local service agent 11 is an agent for furnishing the information and functions proper to the computer 1. The local service agent 11 supervises the local file system 17 to furnish the information to the shell agent 10.

The application agent 12 is of two different types.

Of these, the first type of the application agent is present in the form of wrapping an exterior application 18 and acts as an intermediary when an external application 18 has communication or interaction with an agent in the agent base 9.

The second type application agent is a stand-alone as an agent

in the agent base 9, without having an external application program 18, to execute a specified application.

In any case, the application agent 12 executes a specified task, on request from the shell agent 10.

The shell agent 10, local service agent 11, application agent 12, agent base 9, agent movement module 13, thread generating module 14, remote call module 15 and the agent generating module 16 get respective processings and functions executed by programs run on the computer 11. In an embodiment of the present invention, these programs are read into a computer from a recording medium, having these programs recorded therein, such as a CD-ROM, FD, DVD, magnetic tape or a removable HDD, through a readout device of the recording medium or an interface of the readout device, and an execution image of these programs is loaded in a main memory and executed by the CPU to execute the present invention. Alternatively, the programs, transmitted from a server through a cable or radio communication transmission medium, may be read into the computer through a communication device or its interface.

Fig. 3 shows an illustrative structure of the shell agent 10 in an embodiment of the present invention. Referring to Fig. 3, the shell agent 10 includes, as main constituent elements, a shell interpreter 20, which includes syntax analyzing module (syntax analyzer) 21, receiving an input of a script language from the input module 7 and analyzing the syntax of the input script language, and command analyzing and executing module 22 for analyzing and

executing a command.

In analyzing the syntax of the script language, the syntax analyzing module 21 references a keyword table 27 in which are managed and registered keywords, as a keyword letter string, such as "parallel".

On reception of an input having a syntax beginning with "parallel", the syntax analyzing module 21 requests the thread generating module (thread generator) 14 to generate a new thread.

The command analyzing and executing module 22 references an internal command table 28 to verify whether or not the input command can be processed within the shell agent 10.

If, as a result of verification, the input command is the internal command, the command definition is loaded from an internal command definition module 29 for execution.

If, in executing the command for directory migration, among the internal commands loaded from the internal command definition module 29, the shell agent 10 itself needs to be executed, the command analyzing and executing module 22 invokes the agent movement module 13.

Moreover, the command analyzing and executing module 22 exploits the remote call module 15 in executing methods of other agents.

The results of execution by the command analyzing and executing module 22 are delivered to the output module 8 and are output from the computer 1.

On the other hand, the external command is executed by external command executor module 30. By the agent repository 19, the external command executor module 30 retrieves an agent program which executes the external command. For determining which path in the agent repository 19 is to be retrieved at this time, a repository path table 31 is referenced to sequentially retrieve the agent repository 19 as to paths stored in a repository path table 31.

If, as a result of retrieval of the agent repository 19, the required agent has been retrieved, the external command executor module 30 requests the agent generating module 16 to generate an agent. The agent generating module 16 is responsive to this request to generate the application agent 12.

If the external command executor module 30 is to perform a method call to an agent other than the shell agent 10, the external command executor module 30 also invokes the remote call module 15.

The shell agent 10 includes a status table 23 for managing the state of execution of the shell agent 10, a current directory 24 of the shell agent 10, a shell variable management table 25, an agent referencing table 26 for supervising a variable indicating an external agent, these tables assisting in the execution of a shell interpreter 20.

The shell variable management table 25 is a management table for facilitating statement of the script language by the shell variable. Meanwhile, the shell variable management table 25 may be omitted in the configuration of the shell agent 10 shown in Fig. 3.

Fig. 4 shows a modification of the present invention in which elements of the shell agent 10 are configured differently from those shown in Fig. 3. In the configuration shown in Fig. 3, the external command executor module 30 directly invoke the agent generating module 16. Alternatively, this function can be entrusted to the local service agent 11.

In this case, this local service agent 11 receives the name of an external command and the bus information stored in the repository path table 31 from the external command executor module 30 to effect retrieval of the agent repository 19 and actual invoking of the agent generating module 16 to generate the application agent 12.

The processing and the function of the syntax analyzing module 21 and the command analyzing and executing module 22 of the shell interpreter 20 of the shell agent 10 and the external command executor module 30 are realized by a program executed on the computer 1.

Fig. 5 shows an example of a structure of a storage table for the information responsible for supervising the directories inclusive of the current directory 24.

Referring to Fig. 5, the directory is supervised by pairs of attribute names 32 and values 33. The attributes are comprised of a root base 34, representing an agent base 9 where the shell agent 10 is booted, a current base 35 where the directory exists, and an in-base directory 36 specifying a local directory in the base.

For example, if the shell started in a base A indicates a

directory [/home] of base B, the root base stores the base of the base A, the current base stores the base of the base B, and the in-base directory stores [/home].

In designating the directory by a user, an absolute path may be specified by

(1) [//base name A/in-base directory A] or

(2) [///in-base directory B].

In case of (1), the current base is [base name A], whilst the in-base directory is [in-base directory A]. In case of (2), the current base is [root base], whilst the in-base directory is [in-base directory B].

A relative directory may also be represented by [../], indicating a directory of an relatively higher hierarchy above the current directory or [dirA/dirB] indicating a directory [dirB] below the directory [dirA] below the current directory.

Fig. 6 shows an example of a structure of the status table 23 in an embodiment of the present invention. Referring to Fig. 6, the status is supervised by pairs of the attribute names 37 and values 38, with the attributes being comprised of a parallel executing counter 39, indicating whether or not the shell agent 10 is under execution in parallel, and a terminal mode flag 40, indicating whether the shell agent 10 is executed in the terminal input mode or in the batch mode.

Fig. 7 shows an illustrative structure of a shell variable management table 25 in an embodiment of the present invention.

Referring to Fig. 7, the shell variable management table 25 supervises the shell variables, with a shell variable name 41 and a value 42, paired together, and has a number of variable management units 43 equal to the number of the shell variables.

Fig. 8 shows an illustrative structure of an agent referencing table 26 of the shell agent 10 configured for managing agent variables as variables indicating the agents. Referring to Fig. 8, the agent referencing table 26 supervises the agent variables with an agent variable name 44 and an external agent 45 and has a number of variable management units 46 equal to the number of agent variables.

Fig. 9 is a flow diagram for illustrating the operation of shell interpreter 20 in an embodiment of the present invention.

First, the shell interpreter 20 receives a command input from the input module 7 (step 52).

It is then verified whether or not this command is an end command (step 53). If the command is the end command, the end processing is carried out (step 57) to terminate the shell agent 10.

If the command is not the end command, the syntax analysis is performed by the syntax analyzing module (syntax analyzer) 21, using the keyword table 27, to check whether or not the sentence being analyzed is a parallel sentence (step 54).

If the sentence is the parallel sentence, the shell agent 10 is to execute commands in parallel. So, the shell interpreter 20 proceeds to parallel execution processing of step 58.

If the sentence is not the parallel sentence, the shell

interpreter 20 verifies, by the command analyzing and executing module 22, whether or not the sentence can be processed as an internal command using the internal command table 28.

If the command is the internal command, the shell interpreter 20 proceeds to the internal command execution at step 59 and executes the internal command using the internal command definition module 29.

If the command is not the internal command, the shell interpreter 20 proceeds to external command execution (step 56).

If the parallel execution processing (step 58), internal command execution (step 59) and the external command execution (step 56) have come to an end, the shell interpreter 20 returns to the command inputting processing of step 52 to continue the processing.

The step of verifying whether or not the command is the end command of Fig. 9 (step 53) and the processing of verifying whether or not a sentence in question is a parallel sentence, by syntax analysis, may be reversed relative to each other.

Fig. 10 is a flow diagram in an embodiment of the present invention for illustrating the processing sequence of the internal command processing in the command analyzing and executing module 22 of the shell interpreter 20 of the shell agent 10 (step 59 of Fig. 9).

It is first verified whether or not the internal command is a command [cd] (cd command) indicating directory migration (step 67).

A command other than the [cd] command is executed in accordance with the internal command definition stored in the internal command

definition module 29 (step 73).

If the internal command is [cd], the directory of the destination of migration 47 is extracted from the cd command, for which the direction of the destination of migration is now specified.

In the present embodiment, the user extends the directory into directory constituent elements, shown in Fig. 5, in order to utilize the directory in the form of a [//base name/in-base directory], and sores the base name and the directory in the base in the current base 35 and in the in-base directory 36, respectively.

The shell interpreter 20 then checks whether or not the value of the parallel executing counter 39 provided in the status table 23 is equal to [0] (step 68).

In this parallel execution counter, an information on the depth of parallel execution is stored as an information representing whether or not the shell agent is currently performing multi-thread operations.

If the parallel execution is not running, the value of the parallel execution counter is [0]. If the shell agent 10 is in the simplex parallel syntax, or in the parallel syntax, the value execution counter is [1] or [2], respectively.

If the value of the parallel execution counter is other than [0], it indicates that the shell agent is in parallel execution. Since the effect of migrating the entire shell agent to some other computer is small, only the operation of updating the value of the current directory 24 to the value of the directory of the destination of the

migration indicated by the cd command (step 72), with the shell agent itself not being migrated between computers.

If the value of the parallel execution counter is [0], the shell agent is executing at a sole thread. Thus, the terminal mode is inspected (step 69), next.

A terminal mode flag 40, provided in the status table 23, has its value set to [1] or to [0] when the shell agent is being executed as a shell awaiting an input from a terminal or when the shell agent 10 is being executed as a batch command without being connected to a specified terminal, respectively.

If the value of the terminal mode flag 40 is [1], the shell agent is awaiting a terminal input, so that, if the shell agent has moved to another computer, the input response rate is lowered. Therefore, movement of the shell agent by the cd command between computers is not done and the program branches to step 72 where only the operation of updating the value of the current directory is performed.

If the value of the terminal mode flag 40 is [0], the shell agent 10 is being executed in a batch mode. It is more advantageous for the shell agent 10 to travel to a computer having a working directory. Therefore, the value of the current directory 24 is compared to that of a directory of destination of immigration 47 (step 70).

If the current directory 24 and the directory of destination of migration 47 are on the same computer, that is if the values of the current bases 35 are equal, shell agent movement between different computers is not needed. Therefore, the program branches to step 72

to execute only the updating of value of the current directory.

If the current directory 24 and the directory of the destination of migration 47 are not on the same computer, that is if the values of the current bases 35 are not equal, the shell agent is moved to an agent base of a computer where there exists the current base of the directory of destination of migration.

In this manner, the internal command [cd], issued by a user, is used not only for executing the processing pertinent to directory migration in the computer, but also for selecting the computer configured for executing the shell agent within the shell agent 10.

By this structure, execution of an application on a remote computer can be caused to appear to the user as if a directory has been migrated and an agent is executed, so that, when plural remote application are operated in concert in order to execute services distributed over a network, statement by simplified script language is possible.

Figs. 12 and 13 illustrate changes in the configuration in case of shell agent migration, according to an embodiment of the present invention.

Referring first to Fig. 12, the shell agent 10, situated on an agent platform A 2A, requests an agent movement module (agent mover) 13A to move the shell agent 10 itself.

The agent mover 13A transfers the state of the shell agent 10 to an agent mover 13B of an agent platform B 2B.

Referring to Fig. 13, the agent mover 13B re-constructs the

shell agent 10 on the agent platform B 2B to re-start the execution.

Fig. 14 illustrates a case in which an internal command issues a request for method execution or migration to an external application agent 12 when the internal command is executed in the command analyzing and executing module 22 according to an embodiment of the present invention.

For identifying an external application agent indicated by the agent variable, the command analyzing and executing module 22 acquires a pointer to an external application agent, from the name of the agent variable, using the agent referencing table 26.

The command analyzing and executing module 22 requests the remote call module 15 to execute the method or migration to the external application agent 12.

The application agent 12 has, as remote control accepting module, migration accepting module 48 and command accepting module 49.

After confirming the rights to make request for movement, the migration accepting module 48 requests the agent mover 13 to move the application agent 12 itself.

After confirming the rights to execute the command, the command accepting module 49 executes command executing module 50.

Figs. 15 and 16 illustrate changes in the configuration in an embodiment of the present invention when an application agent 12 in the same computer from the shell agent 10 is moved to another computer.

Referring first to Fig. 15, the shell agent 10 transfers a movement command to the application agent 12, using remote call module 15.

The application agent 12 asks the agent mover 13A to move the application agent 12 itself, whereupon the agent mover 13A transfers the status of the application agent to the agent mover 13B of the agent platform B 2B.

Referring to Fig. 16, the agent mover 13B re-constructs the application agent 12 on the agent platform B 2B to re-initiate the execution.

In Figs. 15 and 16, the application agent lying in the same first agent platform as that where the shell agent 10 lies. Alternatively, an application agent maybe present in a second agent platform different from that where the shell agent lies, with the application agent then being moved to a third agent platform or to a first agent platform where the shell agent lies.

Fig. 17 is a flowchart in an embodiment of the present invention in which a parallel sentence is detected by the syntax analyzing module 21 to execute the parallel execution processing (step 58) of Fig. 9.

First, the parallel executing counter 39 of the status table 23 is incremented by 1 at step 60. This records that the shell agent is currently by parallel execution.

Then, as many threads as there are commands in the parallel sentence then are generated (step 66). It is the thread generating

module (thread generator) 14 that is responsible for thread generation.

The number of the generated threads is set in the variable [threads] (step 62).

In actuality, the commands in the parallel sentence are executed in parallel by each thread.

When each thread completes its own execution, the value of the thread variables is decremented by [1] (step 64). This processing needs to be exclusively controlled between the threads.

It is then verified whether or not the thread variable has become equal to [0]. If the thread variable is not equal to 0, the program waits for the end of the next thread.

When the entire threads have come to a close, the value of the thread variable is [0], at which time the parallel execution comes to a close.

The value of the parallel executing counter 39 is decremented by one (subtraction) at step 66 to terminate the execution of the parallel execution in its entirety.

Fig. 18 shows the configuration in an embodiment of the present invention when in particular a thread generated in the parallel sentence executes a command on another agent platform.

Referring to Fig. 18, the shell agent 10 asks the thread generating module 14 to generate a new thread, after which the shell agent 10 prepares a sub-shell agent 51 on the agent base 9B of the agent platform 2B, in order to execute the internal command of the

parallel sentence.

The sub-shell agent 51 then is able to execute the method or the movement command to the application agent 12 or to the local service agent B 11B.

Fig. 19 illustrates the case of parallel command execution on plural agent platforms in the parallel agent platforms in the parallel sentence.

Referring to Fig. 19, a sub-shell agent 51B and a sub-shell agent 51C are generated by a remote call from the shell agent 10 to enable sub-shell agents to be executed in parallel on the two agent platforms, that is agent platform 2B and agent platform 2C..

Fig. 20 is a flow diagram showing the processing sequence for executing external command processing of Fig. 9 (step 56) in an embodiment of the present invention.

Referring to Fig. 20, a method in which the external command executor 30 directly invokes the agent generator 16, as shown in Fig. 3, is used as a method for starting the external application agent.

It is first checked whether or not the current base stored in the current directory 24 coincides with the currently operating base which is actually executing the shell agent (step 74).

If the result of check is affirmative, it is first verified whether or not the program of the application agent in question is present on an agent repository path of the agent repository 19 of the currently executed base (step 75).

If there is present the program of the application agent, the agent generator 16 is requested to load the program of the application agent from the agent repository 19 to execute the application agent 12 (step 76).

If no agent repository 19 is found, the execution results in failure (step 80).

If, in a step 74 when it is verified whether or not the current base stored in the current directory 24 coincides with the base currently executing the shell agent, the result of check indicates non-coincidence, the sub-shell agent 51 is prepared in the current base using remote call module 15 (step 77).

Fig. 21 shows the relation among different constituent elements in this case.

It is checked whether or not the program of the application agent in question is present in the agent repository path of the agent repository 19 of the current base (step 78).

If there is any such program of the application agent, the agent generator 16 is asked to load the program of the application agent from the agent repository 19 to execute the application agent 12 (step 79).

If no agent repository is found, the execution results in failure (step 80).

Fig. 22 is a flowchart for illustrating the processing sequence for executing the external command processing of Fig. 9 (step 56).

Referring to Fig. 22, a method of entrusting the task pertinent

to the agent generation to the local service agent 11, as shown in Fig. 4, to cause the local service agent 11 to invoke the agent generating module 16, is used as a method of starting the external application agent.

It is first checked whether or not the current base stored in the current directory 24 coincides with the current base actually executing the shell agent (step 81).

In case of coincidence, the local service agent 11 of the current base is asked to execute the external agent (step 82), in order to generate the external application agent on the current base.

The local service agent 11 then checks whether or not there exists the program of the application agent in question on the agent repository path of the agent repository 19 of the current base (step 83).

Should there exist the program of the application agent, the agent generator 16 is asked to load the program of the application agent from the agent repository 19 to execute the application agent 12 (step 84).

If no agent repository is found, the execution results in failure (step 89).

In case of non-coincidence at step 81 of verifying whether or not the current base stored in the current directory 24 with the current base actually executing the shell agent, the local service agent 11 of the current base is asked to execute the external agent, using the remote call module 15 (step 86).

Fig. 23 shows the relation among the different constituent elements in this case.

Referring to Fig. 23, the local service agent 11B verifies whether or not the program of the application agent in question exists on the agent repository path of the agent repository 19 of the current base (step 87 of Fig. 22).

Should there exist the program of the application agent, the agent generator 16 is asked to load the program of the application agent from the agent repository 19 to execute the program (step 88 of Fig. 22).

If no agent repository is found, the execution results in failure (step 89) of Fig. 22).

Fig. 24 shows the relation among the application agent 12, a specified file 96 and an external application 18 booted by the application agent.

The application agent 12, intimately related with a specified external application, is able to interpret and execute the file 96.

Fig. 25 shows the configuration in an embodiment of the present invention for computer-dependent startup of an application 12 employing a general-purpose application agent 97.

Referring to Fig. 25, general-purpose application agent 97 checks an extension of the designated file 96 to reference an extension application accommodating table 98 to determine the application to be executed. This extension application accommodating table 98 is a table which has stored therein extensions

99 and applications 100 in paired state and hence has stored therein the extension-based information 101 per extension by extension.

The general-purpose application agent 97 retrieves items having an extension 99 coincident with the extension of the file 96 to start the application of an application column 100 of a pertinent item.

The extension application accommodating table 98 may be provided on each computer, so that an image file having the same extension may be displayed on a different computer using a different application.

For more specifically explaining the above-described, several specified embodiments of the present invention will be explained in detail.

Fig. 26 shows an illustrative structure of a system configuration embodying the present invention, and specifically shows the configuration of a system sequentially controlling two presentation tools on two computers.

First, there is a shell agent 10A on the agent base 9a of the agent platform 2A of a computer A 1A. It is to this shell agent 10A that an execution script file 92 is input.

In this case, the execution is in the batch mode, so the terminal mode flag 40 of the status table 23 is 0. Since any parallel sentence indicating parallel execution is not used, the shell agent moves through the inside of the computer by a cd command in the script.

By the [cd] in row 1 of a script file 92, the shell agent 10A moves to a local directory [/home] of agent base 9B of an agent platform 2B of a computer B 1B.

The [ppt] command in row 2 of the script file 92 is then executed. It is assumed that the ppt command is the name of the application agent B 12B which starts an external presentation tool B 90.

Therefore, the shell agent 10B boots the application agent B 12B which then boots a presentation tool B 90 to start the presentation.

Then, with the cd command on row 3 of the script file, the shell agent 10B moves to a local directory [/home] of an agent base 9C of an agent platform 2C of a computer C 1C.

At row 4, the application agent C 12C is booted to start execution of a presentation tool C 91.

With [cd] at row 5 of the script file 92, the shell agent moves to the computer B to transmit the command [next] for displaying [next page] on the presentation tool B.

At rows 7 and 8, the shell agent moves to the computer C. The presentation tool C also displays next page.

In this manner, the operation of sequentially folding over pages in the presentation tools of plural computers can be described in a sole shell script.

Fig. 27 shows another illustrative system embodying the present invention and specifically shows the configuration controlling two presentation tools on two computers.

The system shown in Fig. 27 differs from the system shown in Fig. 26 in that a parallel sentence is being used in a script file 93.

In the parallel sentence, a sub-shell agent is prepared in each of the computers B and C to respectively control the application agent B 12B and the presentation tool B 90, or the application agent C 12C and the presentation tool C 91 in parallel.

In the present system, in distinction from the system shown in Fig. 26, the application is run in parallel and the sub-agent [parallel] is synchronized in a row [end] at the end of the parallel sentence.

Fig. 28 shows another embodiment of the system of the present invention, and specifically shows a system for controlling a presentation tool 90 and a WWW (World Wide Web) browser 95 in parallel on two computers. Fig. 28 differs from Fig. 27 in that an object of control is an application different from that of Fig. 27.

Using the present embodiment, there can be easily constructed a system configured for displaying the information of relevant WWW pages in parallel and with synchronizing functions on plural computers.

The meritorious effects of the present invention are summarized as follows.

As discussed above, the following meritorious effects are achieved by the present invention.

It is a first effect of the present invention is that execution of plural applications distributed over plural computers can be

controlled using a sole script program. The result is that there can be easily constructed a distributed application of a higher hierarchy order by plural applications acting in concert.

The reason is that, in the present invention, interpretation of the script language is constructed by a shell agent capable of mobility and communicating with remote computer computers, whilst the application agent controlling the application execution can be universally generated and executed on any optional one of the computers.

It is a second meritorious effect of the present invention that the script can be executed not only sequentially but also in parallel, and that the parallel or synchronous execution of the applications distributed over plural computers can be easily described using a script language.

The reason is that, in the present invention, a block termed "parallel" is provided in a sentence grammar element of the script language, and that, for parallel execution of commands in this block, there are provided the functions of generating plural threads and executing the generated thread in the sub-shell agent in a remote computer.

It is a third meritorious effect of the present invention that directory movement between different computers and directory movement in a computer can be realized by a sole directory movement command such that shell agent movement is enabled without using a special command for agent movement.

The reason is that, in the present invention, there is provided a function of supervising the current directories, the computer within which actually resides a shell agent, or the status of the shell agent, within the shell agent, such that, in case of necessity, the shell agent can move between different computers.

It is a fourth meritorious effect of the present invention that, even if there differs an application provided from one computer to another, each application can be executed depending on file contents.

The reason is that, in the present invention, there is provided a function of specifying and executing an application conforming to the computer so that the application agent is able to prepare a general-purpose application agent to render it possible to selectively execute a computer-dependent application in keeping with the extension of a file which is to be executed.

It should be noted that other objects, features and aspects of the present invention will become apparent in the entire disclosure and that modifications may be done without departing the gist and scope of the present invention as disclosed herein and claimed as appended herewith.

Also it should be noted that any combination of the disclosed and/or claimed elements, matters and/or items may fall under the modifications aforementioned.